Exploration of versioning in product lifecycle management software packages and theoretical enhancements

By
Zhu Meng

MSc operation management — OML 2010
Student identity number 0604601

in fulfilment of the requirements for the degree of

Master of Science
in Operations Management and Logistics

Supervisors:
Dr.ir. H.J. Pels, TU/e, IS
Prof. Dr R.J. Kusters, TU/e, IS

TUE. Department Technology Management.
Series Master Theses Operations Management and Logistics

Subject headings: product lifecycle management (PLM/PDM), versioning
Abstract

Background: The versioning in database management has been discussed and developed for decades. When a lot of theories concerned have been introduced, discussed, accepted or even rejected, little common agreement has been reached. The reason of lack of agreement could be either the development of IT technologies or the diversified application of the versioning in different environments. With both the needs from business and production and provision of mature IT technologies, many product lifecycle management (PDM) software packages have been developed and used nowadays. How the PDM software packages regulate the versioning is a topic both for academic and business purpose. In this paper, the versioning mechanism and architecture of the PDM software packages are explored to compare them with existing relevant theories.

Results: Both SolidWorks Enterprise PDM and Teamcenter Express are applying simple versioning mechanism. Versions are ordered based on time based sequence. Checking in/out in both software packages does not violate any theoretical models even though some flexibility is provided improving the user-software interaction. Status management in both packages in principle conforms to the theoretical models.

Conclusions: The exploration tells that the versioning in practice is still far from theoretical models in terms of the academic level. Some rules in versioning is violated in those software packages. Nevertheless, it seems the violation does not impede the application of the software. Moreover, the exploration indicates theoretical models in versioning are still useful in orienting the future of PDM software since space for improvement is still open.
# Table of Contents

1. **Introduction** .................................................................................................................. 1

2. **Research problems and approaches** ............................................................................. 2

3. **Related Work** ................................................................................................................. 4
   3.1 version numbering............................................................................................................. 4
   3.2 Composite object versioning ........................................................................................... 8
   3.3 Status management ......................................................................................................... 10
   3.4 Summary of literature review ......................................................................................... 10

4. **Concepts and Reference Models** .................................................................................. 11
   4.1 Concepts concerned ......................................................................................................... 12
   4.3 Versioning ...................................................................................................................... 14
   4.4 Checking in/out mechanism ......................................................................................... 14
   4.5 Status ............................................................................................................................ 14

5. **SolidWorks Enterprise PDM** ................................................................................ ...... 15
   5.1 Data distribution in SolidWorks Enterprise PDM ............................................................ 16
      5.1.1 File vault in SolidWorks Enterprise PDM ................................................................. 16
      5.1.2 File types in SolidWorks Enterprise PDM ................................................................. 16
      5.1.4 Object shortcut ....................................................................................................... 17
   5.2 Versioning in SolidWorks .............................................................................................. 17
      5.2.1 Version and Revision .............................................................................................. 17
      5.2.2 Evolution of version/revision .................................................................................. 19
      5.2.3 Version Synchronization ......................................................................................... 21
   5.3 Complex object versioning ............................................................................................ 22
   5.4 Checking in/out mechanism ........................................................................................... 23
   5.5 Workflow and status management .............................................................................. 24
   5.6 Summary of SolidWorks Enterprise PDM and enhancements ........................................ 25

6. **Teamcenter Express** .................................................................................................... 28
   6.1 Data distribution in Teamcenter Express ...................................................................... 28
   6.2 Versioning in Teamcenter Express .............................................................................. 29
   6.4 Checking in/out in Teamcenter Express ........................................................................ 31
   6.5 Status management in Teamcenter Express .................................................................. 31

7. **Problem definition reflection** ...................................................................................... 33

8. **Conclusion** .................................................................................................................... 34

References ............................................................................................................................. 36
1. Introduction

Object-oriented database management has been considered and developed by many researchers since 60s in last century [Sklenar, 1997]. At that moment, it was not given a commonly accepted name. In 1985, the term object-oriented database system was first adopted in a scientific literature. Henceforth, the database in which data or information is represented as object is named an object-oriented database. In managing the document and design data in database, versioning the objects is a spot deserving studying and distinguishing since different methods of it have been proposed by the researchers all over the world. There are both differences and consensus among the proposed versioning methods. Basically there are two categories of versioning, linear and network versioning. The linear versioning and network versioning both have their advantages and disadvantages; nevertheless, the versioning mechanism in a PDM software package concerns more about practical business than theoretical refinement. Status management involves workflow transition and accessibilities of versions. It is interesting to know whether status is the only way for representing accessibilities. Checking in/out usually is the only way to create new versions. The object of checking in/out could be either a version itself or a document containing all its versions.

With the development of database management theories, some product data management software packages have been developed to provide assistance in managing product data in business environments to help managing the product data in its life cycle. These software packages adapt different ways of versioning, thus, it is valuable to look into what kinds of versioning methods these software packages adapt, how status and checking in/out mechanism are configured.

Solidworks Enterprise PDM is developed by Dassault Systèmes, S. A. in France. Dassault Systèmes develops sorts of PLM software packages for different types of organizations based on their sizes and employee numbers. For example, PDM Works Workgroup is mainly for organization with less than 5 members, while Solidworks Enterprise PDM can be suitable to much larger organizations. In March 2009, there are one million licenses all over the world [Jeff Ray, 2009]. Solidworks Enterprise PDM is used by some organizations as well, such as Hartness International that is a machinery manufacturer and Trek Bicycle, Inc that is the leading manufacturer of handcrafted bicycles for the professional racing and recreational markets (http://www.solidworks.com/sw/successes/viewsuccess.htm?record=1155). In this assignment, the Solidworks Enterprise PDM package is provided by a Dutch business consultant agency CADMES. CADMES has experiences in applying Solidworks Enterprise PDM locally and is expanding its customers. They are also keen to educationally expand their service and kind to
arrange meeting for my visits in `s-Hertogenbosch. In the argument in this paper, SolidWorks Enterprise PDM sometimes is called SolidWorks for short.

Teamcenter Express is an integrated product life cycle management suite developed by Siemens PLM Software. It is considered as the most popular PLM software with over 3 million licensed users in 2007. Teamcenter Express is applied by famous American Lockheed and its Russian rival Sukhoi as their PLM software. Teamcenter Express is also used by University Racing Team in Technical University Eindhoven for product lifecycle management. Several versions of Teamcenter have been developed. In this paper, the simulation is implemented on Teamcenter Express 3. Also Teamcenter Express is mentioned as TC for short in this paper.

Due to Teamcenter Express’ wide application in industries and SolidWorks Enterprise PDM’s popularity in Europe, these two software packages are chosen to be investigated for their product data management techniques.

2. Research problems and approaches

In data management, adequately versioning the documents and trace the evolution of version propagation are appreciated for the sake of efficient and reliable data management. How to version the data has been a topic for decades so far, however, there are still different theories about it. Thus, it is useful to examine what versioning theories the software packages adapt and whether these theories are effective and robust.

The research problems are divided into the following three research questions.

1. How are the objects in software packages versioned? Do they conform to the theoretical concepts? If not, what could be the reasons for that?

The research question 1 contains two sub questions to further define the research topics about versioning mechanism exploration.

1A. How is the numbering methods of versions implemented in the selected PDM software packages?
Research question 1A is designed to find out the version numbering method in the software packages. The challenge in version numbering concerns the version numbering form or the order to modification. For example, linear versioning and network versioning are both general scope of versioning including specific versioning mechanism. Moreover, there is a concept of revision in some PDM software packages. It is interesting to look into what a revision represent and why it is used in database management.

1B. Is there any method implemented in either Teamcenter Express or SolidWorks Enterprise PDM to trace the evolution of versions?

The traceability of evolution of versions is important when the time sequence and modification history do not coincide. In many scientific articles, version derivation graph (VDG) is used to model the version tree of an object. Nevertheless, when the object is a complex (composite) object, the version propagation is a tough issue again. A new complex object version does not necessarily mean an improvement relative to its predecessors but only a version with different versions of component objects.

Currently linear versioning is used in most cases since its numbering of versions is easier than network versioning. Nevertheless, when time sequence and modification history does not coincide [Hamer&Lepoeter, 1996], the linear versioning does not display clear modification history while time sequence is its focus.

2. How are versions and documents checked in/out in selected software package? Does checking in/out protect the contents from being edited by more than one user?

The documents should be checked in/out to create new versions. Checking out a document blocks this document for checking-out by other users. This limited access setting is a protection to contents of the version. In case a document is checked out for a long time (a couple of days) for some reasons, unnecessary delay of version evolution could be caused. Therefore, when the user wants to share the new content, he needs to check in the new content to publish the new content creating a new version. Thus it is useful to look into whether the software packages can facilitate immediate checking-in when a user has not finished his editing yet but still can share parts of contents.

3. How do SolidWorks Enterprise PDM and Teamcenter Express control the status of the versions, and what do they do with workflow?

Statuses of versions concern the maturity of data. Versions with different statuses are available for the users with different access rights. Usually when a version is approved by the administrators, its content is available to the more users than in other statuses. Status management therefore is related to the approval of a version by an administrator. The change in status causes either promotion or demotion, which are also called workflow transition. Thus, workflow transition sometimes can be regarded as the promotion and demotion of a version even though it is not always true in all situations. The status in a workflow can have something to do
with the actions that a user could perform. Usually an approved version allows more possible actions than a version in other statuses.

We pay our attention to how those functions in theoretical models are realized in inspecting the PDM software packages and why if they are not realized. Also it is possible that some functions are provided in PDM software packages while does not exist in theory at all. Then exploring the reasons of violation against reference models is important to understand the differences in architecture between the theories and practices.

3. Related Work

Versioning mechanisms in all kinds of environments are being explored by many researchers. The researchers’ theories vary very much and little theories developed are oriented for practical business. By reviewing those scientific articles, diversified versioning theories from different researchers in different cultures can be examined and compared with each other to distinguish the pivotal logic behind them. This section consists of two parts, version numbering and status management.

3.1 version numbering

By version numbering, we mean mechanism of assigning numbers to versions. There are different ways of numbering in the scientific articles reviewed. It is useful to compare them in order to seek generally-accepted rules in version numbering.

[Pels et al, 2010] propose two principles. First, there should be clear distinction between the version of the product and the versions of the documents while version numbers should be ordered. This means there are versions for both the products and its documents. Ordered version numbers define that a higher version number represents the improved version. Second, the contents of a version is not changeable, otherwise, new versions are generated. In this way, versions are frozen in terms of its content. This principle applies not only to simple object but also to all object versions.

[Khaddaj et al, 2004] describe the ordered version number rule in both mathematical and graphical terms. Both forward and backward linear versioning are examples of linear versioning. Khaddaj [2004] also explains the branching versioning in some instances Figure 1 graphically shows the structure of branching versioning. This branching versioning is used when there are alternative ways to work out improvements. One generic version is parent to several children versions. However, it is not clearly explained how parent versions are merged. There is critical argument about this branch versioning. This branch versioning is not considered as a versioning mechanism; instead, it is a variant mechanism. In figure 1, V1,0 is parallel to V1,1 and V1,2. These three are all variants evolving from V0. After separate evolution, one of them is chosen for further development.
[Talens et al, 1993] agree with the ordered version number rule by stating that a new version should always be added at the end of the sequence and behind the latest version. In this theory, attributes of the versions orient the versioning. There are class versions and instance versions proposed by Talens [1993]. Class versions are different in both attributes and values, while instance versions share same attributes originating from its class version. The distinction between class version and instance version resembles the distinction between version and variant [Pels, 2010]. A class version is actually the version that is usually called, while the instance version is actually the variant. The variants share some common values of attributes while there are differences between each other. A version number consists of two parts. For example, Va,x (a and x are numbers) represents a version originating from the class version Va,x and is the xth versions originating from Va. When there is a change in the attributes of the versions, class version evolves to Vb. The instance versions originating from Vb are Vb,x. The mechanism of their versioning is shown in figure 2. The version V1.0 and V1.1 are derived from the class version V1, therefore they inherit all the attributes of class V1.
The attributes of a version is open to specific environments. For example, a product’s parts remain unchanged, then the attributes remains unchanged as well. If one more part is added in the assembly, then the attribute is changed. This theory is created mainly for product versioning since document’s attribute is not easy to define.

Not all the version branching is made purposely. In design process, it is necessary sometimes to go back to certain previous version to improve in a different way. [Hamer&Lepoeter, 1996] defines the cause of this branching to be the inconsistency between time sequence and modification history. In figure 3, version 6 is not an improvement to version 5; instead, it is one of the two child versions of version 2. It appears to be a branch versioning, which is actually not the truth. If the versions are ordered based on time sequence, the version derivation cannot be shown. Whether this is critical or serious still depends on the business requirements. In production, it does not cause much inconvenience since it is always the latest version of a design that is used for production. It is worth of noticing how the commercial PLM software packages deal with this situation.
The order of versions is a troublesome issue in versioning since in different versioning mechanisms the order of versions is always related to evolution history (modification history), time sequence, and version coding. When modification sequence is not the same as the time sequence, usually a network of versions arises. The network versioning usually takes place when the engineers want to have different options in developing. The alternatives of developing cause variants. Network versioning imposes catastrophes on version coding, thus brings too much complexity in database management. The rules used as reference in this paper do not bother the consistency of the modification history and time sequence; instead, only improvements of contents (the improvements are not objective sometimes) are concerned.

![Evolution Diagram](image)

**Figure 4 Example of evolving versions from old versions instead of the latest version**

In the figure 4, version 4 is created after version 3; however, version 4 evolves from version 2 instead of version 3. No matter why the version 3 is forbidden to be improved, version 4 can still be regarded as the improvement of version 1, 2 and 3. Thus the evolution of versions from an old version instead of the latest version is still consistent with our reference model of versioning.

[Krishnamurthy&Law, 1997] refer a complex versioning method in their research. The referred method comes from [Keller&Ullman, 1994]. It is a method of versioning in lexicographical order. It is a mix of version hierarchy and product hierarchy. Figure 5 is an example version hierarchy of an entity numbered in lexicographical order. Version b-1’s first successor has version number of b-2 because it shares the same product structure with its ancestor version b-1. The other successor has version number of b-1a0 because its product structure has been changed already. This versioning mechanism allows the users to trace the development of a product design. Krishnamurthy and Law also propose to describe a version as a string formed by concatenating the entity identifier and the number of the version within the derivation hierarchy [Krishnamurthy&Law, 1997]. Version b-1a0 and version b-2 are variants.
composite object versioning

Composite object version management is much more complex than simple object because of its constituent references. Product in PDM context is the abstraction of the product in real world. The number of combinations of a complex object grows exponentially with the number of components and versions [Asklund et al, 1999]. For example, if we have a composite object containing m simple objects (objects without constituent objects), every simple object has n versions, and consequently theoretically we have $n^m$ different configurations. Furthermore, simple objects’ version management may also involve mechanism designed for composite objects. For example, one cylinder used by a machine requires two representations—CAD representation and NC representation [Baldwin&Chung, 1995]. CAD representation shows the physical specification of the object (cylinder) while the NC representation shows the interface between the design and practical manufacturing by specifying the numerically controlled program. Versioning the cylinder and its CAD and NC representation is not concluded in the paper [Baldwin&Chung, 1995]. The cylinder and its two representations are treated as an entity of a complex object. Theoretically the change on any representation or cylinder itself can cause a new version of the cylinder. However, the authors did not conclude what kind of change could cause a new version. One more example is the design of electronics. Sometimes both electronic schematics and net list are required to be used to represent one single object [Gu, 2009].
sense, CAD and net list are in turn simple objects. Therefore, composite object management applies to both composite object and some simple objects with constituent representations if required (cylinder example above). In our discussion following, both of them would be treated as composite objects for the sake of easy reading and understanding. The complexity of composite object versioning is mainly from the binding process, in which a lot of object versions are duplicated resulting in the so-called version propagation or version proliferation. For example, in case a small part in a complex object is changed, theoretically a new version of complex object could be generated. However, in practice, a lot of data remain unchanged between the old version and the latest version. Version proliferation occurs. Therefore, one of the challenges in complex object versioning is to determine what kind of change could trigger a new version.

As composite objects contain components, it deserves management as to which component is compatible with specific components. Dynamic strategy and static strategy are commonly proposed [Asklund et al, 1999] [Hamer&Lepoeter, 1996] [Miles et al, 2000] In some researches, they are called static (specific) binding and dynamic (generic) binding as well [Carnduff&Goonetillake, 2004].

As composite objects contain components, it deserves management as to which component is compatible with specific components. Dynamic strategy and static strategy are commonly proposed [Asklund et al, 1999] [Hamer&Lepoeter, 1996] [Miles et al, 2000] In some researches, they are called static (specific) binding and dynamic (generic) binding as well [Carnduff&Goonetillake, 2004].

Figure 6 Constructing specific configuration versions in an example of dynamic binding (reproduced from [Carnduff&Goonetillake, 2004])

Figure 6 is the diagram illustrating the mechanism of the generic-generic approach of dynamic binding proposed by [Carnduff&Goonetillake, 2004]. The composite object configuration contains two components and one composite version versions of configurations that should be used in different configurations. The advantage of this approach comes from reuse of components versions. In this way, a specific configuration version list can be made in place of version derivation tree to show configurations in temporal order. That means configurations are in fact the versions, the so-called composite object version 1 just records the composition of the configuration. In figure 5, the two configurations consist of the same components but different versions; therefore, they share one same composite object version. However, the temporal order in that list does not indicate derivation relationship any more. Configuration 2 is not derived
from configuration 1. For example, configuration 2 and configuration 1 are both on the list of composite object version 1; however, configuration 2 is not derived from configuration 1 anymore.

3.3 Status management

Almost in all scientific articles the status of a version is defined to be used to distinguish among different purposes of use of this version. Thus, the basic purposes of defining the status are convergent. For example, the status is considered as the criterion for determining the stability or instability of a version [Talens et al, 1993]. Thus, the basic purposes of defining the status are quite similar; however, the ways to define are different.

Because of different understanding and requirements in categorizing the versions, different ways of defining the status are proposed. For example, two statuses—permanent and temporary status are defined by Talens and his colleagues [Talens et al, 1993]. Three statuses—TV (transient version), WV (working version), RV (released version) are proposed by [Santoyridis et al, 1997] [Miles et al, 2000]. Takeshi Kamita proposes a two-layer status management consisting of personal layer and common layer [Kamita, 1994]. The layer termed by Kamita is actually the status in our terms.

In all the proposed status management, status management mechanism does not specify the location where data are stored; instead, it allows free access of users to data through integrating dispersive databases.

3.4 Summary of literature review

The literatures examined are mostly focusing on versioning mechanisms themselves. No matter the versioning mechanisms are designed for products or documents, complexity is their common property since the researchers are less restricted by practical business implementation. Some of the versioning methods are too complex to be realized in business recently. For example, the lexicographical order versioning is good for managing data purely, but the PDM system does not only need to manage the data, but also needs to facilitate the production based on data controlling. Therefore, such a complex versioning mechanism should be out of the scope of our discussion. It seems the researchers are interested into developing sorts of complex and precise version coding mechanisms to satisfy the business requirements. For example, the lexicographical versioning is a nice solution in terms of industrial technology; however, in practice its implementation might be even more difficult than managing the data.

There are two reasons for versioning. First is the requirement of ordering versions in the PDM context. Usually the latest created version is considered as an improvement of the earlier ones. Thus, compared with other elements, time sequence is considered as the most relevant in identifying the latest improved version. Even though there are some other versioning methods
intended for more precise and complete versioning, it is sufficient to identify a needed version with the simple time sequence versioning. Second is that sometimes the engineers need to go back to a previous version to restore the designing because the design goes in a wrong way. Therefore, it is possible to have several versions developed in parallel. Nevertheless, this should not be in the scope of PDM system.

Note: I think there are two different reasons for versioning. In the PDM context the reason is that objects evolve over time and versions represent their status as communicated at different points in time. From this point of view there is an intrinsic time order. Because of the evolution supposes improvement the later version is always considered an improvement of the earlier. Although many more or less complex version identification schemes are use, It is sufficient to identify versions by a simple sequence number.

For engineers in the organization of their work there is a second reason for versioning. Since a change may later on appear to be less than an improvement, the engineer needs to be able to track back to the latest version before introduction of this error. Also he may want to be able to compare alternative design choices and therefore maintain sequences of versions in parallel. This however should stay inside the workspace of the engineer and needs not be made visible to other parties in the design process. Therefore this is not a function for the PDM system, but should be left to the authoring system (CAD, Word etc.).

Status management in the literatures is mostly about maturity of versions within an organization. Different researchers propose several different kinds of status coding. The difference of these status coding is mostly about different classification of maturity of data in the theories. For example, Takeshi Kamita [1994] proposes a two-layer architecture consisting of personal layer and common layer while most of other researchers prefer more layers [Kamita et al, 1994].

There is not much discussion about checking in/out in the literatures. Therefore, we would like to establish a reference model about checking in/out in order to compare the software packages with it.

4. Concepts and Reference Models

Data management is a topic filled with lots of opinions, models, methods and concepts. Some concepts proposed by different users are not compatible with each other. (H.J.Pels et al) collect some rules in data management agreed by most of researchers and developers. In this paper, these concepts will be used as the reference and to establish the reference models. For the complex object, there is little consensus in field at present, thus, there is no reference model of complex objects’ versioning that is used for comparison. Instead, a promising complex objects’ versioning model and a practical versioning method will be discussed.
4.1 Concepts concerned

Concerning the data management, especially the version management, there are many terminologies that are not unified yet. In order to avoid any confusion when referring to concepts in this paper, the relevant concepts and terminologies are introduced first.

Object: the information system uses objects to represent entities in the design world. An object has attributes to represent properties of the entity.

Product: A product is the physical result of a practical design process, a mapping from an abstract functional description to tangible entities. [Gorti et al, 1997]. The concept developed by Gorti [1997] is not from the viewpoint of PDM. Therefore, In the PDM context, we define that a product is a class of similar product instances that conform to the same specification.

Document: A document is any physical or symbolic sign preserved or recorded intended to represent, or to reconstruct or to demonstrate a physical or conceptual phenomenon [Briet. Suzanne, 1951]. Simply it could be restated as textual record [Buckland, 1997]. In the context of PDM, we could define that documents are the descriptive content of a task. The document is the abstraction of the document object which is actually the version. A document evolves in a number of versions. It is the main deliverable in a design task.

Complex (composite) entity: In most scientific articles, complex or composite entity mean a product consisting of parts. In other words, complex entities can be disassembled into smaller entities. In some PDM software packages, the term complex entity has other names, such as assembly in SolidWorks Enterprise PDM. In this paper, complex object, composite object and assembly are used interchangeably as well. In the literature referred in this paper, there is only complex product, and no complex document. The part as a complex product is products as well.

Version: A version refers to a particular state of an evolving entity. Therefore the versions of an entity are ordered in time. Products and Documents represent evolving entities and therefore they may have Version objects in the database.

Revision: Revision is applied in some PDM software to represent the versions. Thus, it is actually another term of version with its special purpose of use.

Status: The status of a version is the criterion of determining the stability or instability of a version [Talens et al, 1993]. The concept proposed by Talens [1993] deviates little from the commonly agreed concepts about the status. The status of a version indicates the version’s maturity. Status is named based on most scientific articles. However, the status is not always called status. For example, in SolidWorks Enterprise PDM, state means the status. In order to unify the terminologies in this paper, status is named to represent a version’s maturity while state represents a version is checked out or checked in.

State: A version’s state is either checked in or checked out. State is an attribute of Document. Upon check-out the latest version is copied to the editors work space. Upon check in the new content is copied from the workspace into a new version.
**Check in/out**: Checking-In is the only way to create a new version. A document can only be checked in when its state is Checked-Out. In principle a document must be checked in by the same user who checked it out. Therefore a document is checked-out to a particular user. However exceptions to this rule must be possible.

**Copy and reference**: PDM systems support the single source principle, meaning that there is only one single ‘source’ copy of the contents of each version. In the database there is only one version object with one value for the content. In order to maintain relationships a version can be referenced by many different objects in the database. However creating a reference does not create a copy of the contents.

Copies of versions can be created for two purposes: (1) for speed of access a physical copy can be maintained by the system in another location, but this should not be visible to the user, and (2) a copy can be taken form of a version to create a new version for another object. This can be applied as a form of reuse. Further a copy of a version is exported to the personal workspace of a user upon check-out or upon a download action. From that moment this copy is out of control of the PDM system and what happens between a check-out and download and a check-in is completely the responsibility of the user.

### 4.2 Data in database management

In this section the theoretical concepts are specified and explained in a UML static structure. Figure 7 shows how documents and products are represented by different object classes. Document defines the common attributes that is shared by its instances DocVersions. The ProdVersion inherits the attributes from the class Product. Nevertheless, the situation in real world is much more complex for products than documents. Usually the Product class defines the common attributes that are shared by its instances. A problem is that its instances usually could be not defined as an object but a class as well [Pels, 2006]. An attempt by Pels [2006] is that based on the attributes shared classes could be members of each other in order to model the products’ relationship more natural.

![Figure 7 Relationship between document and product](image)
4.3 Versioning

Versioning is the data management concerning the evolution of the versions of a design. According to [Pels et al, 2008], versions obey the following rules:

- The VersionNr specifies the time order of the versions of a document. Higher numbers always represent improvements over its predecessors.
- Without a new version created, a version is not allowed to change.

4.4 Checking in/out mechanism

The Checking in/out is the only way to create a new version. In order to edit a version, the document should be checked out at first. The version is checked out to the local work space of the user. The checked out version is not available for other users to edit but only to view. After editing, the document is checked in creating a new version. Therefore, checking in/out is also a protection mechanism. Usually, only the person who checked out can check in, however, there must be exception as discussed above.

4.5 Status

Status management concerns the description of the life cycle of the versions. The theories concerning status are:

- Each version has its own status,
- status indicates allowed actions

Status codes represent the life cycle of a version. The codes should be used to show the different purpose of use of the version and the maturity of the data in the version.
As shown in figure 8, the status of a version is specified by its lifecycle phase. Promotion is the way to change the status. Each version is in one status in a certain period until this version is promoted. Status can be promoted and demoted; however, the demotion is not recommended in most cases. In case this version is used to create the next version, its content is checked out to be used as the basis of the next version. Thus, the original version’s status remains the same.

DocVersion are modeled as instances of Documents. This implies that Documents seen as classifications (abstractions) of their Versions. The Docversions inherit the attributes from their Document, such as DocId, Title, Author, etc. The Promotion class represents the promotion history of the versions. PromotionTime and the promoter of the document versions are recorded. Checking in/out is used to create a new document version.

The reference models established in this chapter will be compared with the architectures extracted from the software packages to discover the fitness of reference models and find out why if there is no match between each other. The focus of comparison should be on the versioning mechanism because there is a big difference expected between theories and practical implementation. Since the status management does not vary much in different scientific articles, it is expected to have similarity between the theories and practical use.

5. SolidWorks Enterprise PDM

In the discussion of SolidWorks Enterprise PDM, the distinction between state and status in the reference model should be temporarily ignored since in SolidWorks Enterprise PDM statuses are called states. In order to eliminate any potential confusion, we take the commonly agreed definition. Therefore, we still term the state in SolidWorks Enterprise PDM status.
5.1 Data distribution in SolidWorks Enterprise PDM

All data in SolidWorks is represented as documents with different extensions. There is not really a distinction between the product and the document; nevertheless, the product is shown as documents in SolidWorks. Even though the documents representing products and products can be distinguished from their extensions, they are treated as documents in version evolution. Therefore, in order to eliminate any confusion of data type, we use object to mention both the product and the document. All data is stored within a database created when installing SolidWorks. This database is called file vault in SolidWorks, and could have several local file vaults.

5.1.1 File vault in SolidWorks Enterprise PDM

All objects are stored in the file vaults created within SolidWorks. The file vault is the location where data is stored in the database of SolidWorks. The basic structure is that there is one central file vault And multiple local file vaults. The central file vault is available to all users, while the local file vaults is only available to the local users. A user can see all data in central file vault and the local files in his local file vault. Checking out the data from central file vault allows a user to edit the data in his local file vault because checking out copies the data from central file vault to local ones. In local file vault, data could accumulate gradually with the operations checking out, but the local user can not know how much data is in his local file vault. That triggers data redundancy.

An object does not show visibly unless it has versions: a document is created by storing its first version. Visible contents of an object are its versions. In case of SolidWorks, the revisions are visible contents as well. In the interface of SolidWorks, the users see the name of the document, its latest version’s size, the user that checks out it if applicable, etc.

5.1.2 File types in SolidWorks Enterprise PDM

In SolidWorks, a lot of object types are available, such as MS office formats, images (BMP, JPEG), SolidWorks types (drs, prt, sod, etc) and other Object types.

The products in database management still need their carriers---documents. In this paper, the products are treated as synonyms both representing products and parts and we take the term product in our argument. When managing those data, users are still dealing with documents as the carriers of products In SolidWorks Enterprise PDM, the versioning does not vary between the product and document since products are represented as document. Therefore, in our discussion we only discuss the versioning of documents and we use object as mentioning both

### 5.1.4 Object shortcut

In SolidWorks, users are allowed to create shortcuts for some documents in their specified locations. In Figure 9, the document with little yellow cross is a shortcut of its original instance. This shortcut is created by copying the original instance followed by pasted shared after right click. It is like referring an object within the file vault. Actually there is only one original object, but many shortcuts can be created. The change made to any shortcut would be saved to its original instance. In practice, some companies adapt this function when they create a folder for a project to get shortcut of many drawings from library.

![Figure 9 An example of object shortcut](image)

### 5.2 Versioning in SolidWorks

In SolidWorks, versions are also used to improve designs, share ideas, etc. The time sequence ordered propagation of versions in SolidWorks offers convenience to management while the revision concept makes it practical to keep the data from the outside environment. As an object-oriented database management system, product and document are different types of objects in SolidWorks. In order to eliminate unnecessary confusion, object is used to name all kinds of instances in SolidWorks.

#### 5.2.1 Version and Revision

In SolidWorks, versions include two terms----versions and revisions. Version and revision are not synonyms. Both version and revision are visible content of an object. Versions are created and improved within a specific group of the organization, such as a subgroup of the R&D department, a project team, etc. Only the users within the specific group (usually the group developing the versions) or some super users with administration permission have access to the contents of versions created within the group.

In SolidWorks Enterprise PDM, there is the version for review within the organization. Since there is no distinction between product and document in SolidWorks Enterprise PDM, all the discussion about versioning in this part is document versioning. A revision is not only a kind of version, but with different purpose of use.
Figure 10 represents the structure between object and their versions. A revision is a special kind of version. In UML diagram, the revision is a specialization of its version. The revision as a class in UML diagram has only revision number as the attribute.

Assigning revision numbers to versions has two ways, changing the statuses of the versions or increment revision numbers directly. Approval of a version creates both a new version and a revision with the same contents. Incrementing revision creates new revisions only. It is allowed in SolidWorks to create new revisions without change in contents by incrementing revision. The incrementing revision function concerns the user defined revision number in a workflow state. The function incrementing revision will be discussed in next paragraph with the user-defined revision number assignment in the workflow transition.

When several versions with same states are used to create revisions for external review, the user-defined revision numbering scheme functions. The versions evolve in the workflow. When creating a revision number to a version, the status of this version decides what kind of revision number could be assigned. The consequence is that there are revisions with different revision numbering schemes. In figure 11, two status specific revision numbering schemes are shown. In order to create several revisions based on the versions in one same state, the users need to increment revision manually. This is the second way of creating a revision. SolidWorks Enterprise PDM provides this function with risks. For example, a serious problem is that the reviewer can hardly find out which revision is the latest if these revisions are not ordered in time sequence.

Certainly users can also choose to keep one single revision numbering scheme. Actually the default revision numbering scheme is alphabet.
If a version has a revision number, it can have two identities, version identity and revision identity. Both versions and revision in SolidWorks are ordered with alphanumeric version numbers as attributes based on their time sequence of creation. A revision number can consist of static text (optional) and one or more counters, which is known as alphanumerical (Administration Tool. Dassault Systemes SolidWorks Corp). Users are free to set up revision numbers to adapt to their numbering scheme. Examples of revision numbers are:

2. A01, A02, A03, .......
3. 1.1, 1.2, 1.3, .......

5.2.2 Evolution of version/revision

In SolidWorks, the evolution of version/revision is time sequence based. Checking out is required before a version could be edited in order to create a new version. The figure 12 shows the process of creating a new version. In this process, it is noticeable that just saving the change does not create a version. The saved contents are still in the users local file vault, not in the public file vault. Without checking out, a document is read-only. Double clicking a file just open it in read-only mode, users can only review it.
The relationship between revision creation and approval status deserves careful discussion. An approval of a version triggers new revision creation automatically. A revision creation does not necessarily implicate a change in workflow state or a transition of workflow. By simply incrementing revision (right click a document and then increment revision), new revisions can be created even if there is no change in content or status. On the other hand, once a version is approved, new revision is automatically created due to transition of workflow. In figure 13, example of revision and version is given. Revision A is the synonym of version 5, while version 6 is the synonyms of revision B. Revision B, C, D and E have the same contents, which means revision creation does not require content change as a premise. This is a dangerous allowance by SolidWorks Enterprise PDM. The revision B, C, D and E are still recognized by the system as different document even though there is no contents change among. In practice, this phenomenon could be avoided by the users’ careful operation.

![Figure 13 Version and Revision in SolidWorks](image)

When a revision is supposed to be created to share the version content with others, users are allowed to change the status (state in SolidWorks) into “no approval required” to trigger a workflow transition in order to create a revision as well. However, in SolidWorks the version’s status is still marked as approved after being treated in this way the “no approval required” status brings both convenience and danger to data management. In case there is a non approved version, ‘no approval required’ will result in state ‘approved’. Nevertheless, such acceleration skips the inspection procedure of the revised content, thus, it might bring mistake in version evolution. The further danger is that once a revision is created anyway, it cannot be fixed or deleted to correct the mistake unless creating next revision.
The label function (text following the version numbers) allows users to briefly describe the version. Same label content can be used on many versions, even though versions exist in different folders. The label is not an attribute of the version. When a short cut is established for a version, the label of this version is not in the shortcut.

5.2.3 Version Synchronization

The difference between a version in a user’s local file vault and a version of the same object in the central file vaults often brings inefficiency in management (R.A.Baldwin&M.J.Chung, 1995). It is optimal to always work on the latest version of an object. In figure 14, the interface of SolidWorks is clear. The names of the documents (object) do not change with the evolution of their versions. The size of the document (object) is always the latest version’s size. The state here is actually the status in the reference model, where status and state are defined.

![Figure 14 “Object-oriented” interface in SolidWorks](image)

In SolidWorks, users will see a warning if the version on their local file vault is not the latest. Users can respond to get the latest version (right click on the document, then get the latest version). This warning is just used to remind the users to be aware of the new versions. In case a user neglects the warning, he would have to work on the latest version of the object since checking out the latest version is the default checking out. Figure 15 is the real interface in SolidWorks Enterprise PDM when a non-latest version is on the local space of the users.
Figure 15 Version synchronization warning in SolidWorks local file vault

In SolidWorks, users are allowed to review and edit old versions. In order to edit the old versions, the users need to first check out the object, then select the desired old version by clicking get version. Then the users are allowed to work on the version evolution from an old version. This procedure of retrieving old versions reveals that checking out in SolidWorks does not point to one specific version, instead, it checks out the entire object with all its versions. After checking out, the users can choose the one they want to edit. After editing and saving he can create the new version by checking-in. In figure 16, the process of creating a new version from an old version instead of the latest version is shown.

Figure 16 Old version’s checking out

5.3 Complex object versioning

Complex objects are also called composite objects. In SolidWorks Enterprise PDM, complex objects are called assemblies. Complex objects are also represented as documents in SolidWorks Enterprise PDM; however, they are products since documents do not have parts to be assembled. The products in SolidWorks Enterprise PDM have references to their parts. The extensions of products can also tell that they are actually representing products. Complex objects are also versioned based on time sequence in SolidWorks Enterprise PDM. The versioning mechanism is simple but insufficient to provide enough information about the evolution of parts in the object.

In SolidWorks Enterprise PDM, products usually have the extension of ‘sldprt’. The products and its parts are represented as documents. The evolution of the product version is like the
evolution of the document. In order to evolve the product version, the product (document representing the product) should be checked out and checked in. The parts of the product evolve independently from the products they belong to. When there is a change in a part, and the change is approved, the product to which this part belongs does not evolve accordingly.

The products’ versions in SolidWorks Enterprise PDM only increment when the configurations of this products change. By configuration, usually it means the position of the parts in the product. Only when the configurations of the products change, the new versions could be created. Thus, the complex object versioning in SolidWorks Enterprise PDM does not conform to any theories explored in related work; instead, it is just a versioning like document. When a product is checked out, the user can choose to check out the product with latest versions or check out the product with its original design (parts with their first versions).

### 5.4 Checking in/out mechanism

The checking in and checking out in SolidWorks Enterprise PDM is the only way to create new versions. They can easily be done by clicking the check in/out button on the tool menu or through a right click on the object. Once a version is checked out by one user, other users cannot check-out the same document. When a user wants to edit a document, he first of all needs to check out the document into the local file vault. One distinction between saving the documents and checking in is that saving documents does not check in the documents. Users only save the revised contents in its local file vault.

The read-only access applies also when several instances of one same document exist in different folders in one same file vault, which means once one instance of the document in any folder is checked out by one user; all other instances of the same document are not available for editing until it is checked in. When checking in, users need to check in the *originally checked out instance* of the object, which prevents the editing of the object by other users? For example, instance A of the document is checked out. Then instance B of the same document is read-only to all other users. When saving the change and checking in, the user must check instance A into the file vault.

SolidWorks provides a function of allowing versions to be read by other uses while the user checking out the version can keep on editing. In figure 17, the difference between two checking-in mechanisms are shown. By keeping a document checked out, a new version is created and checked into the file vault. The user is still keeping a checked out version that share the same content with the newly created version. Essentially keeping checked out equals checking in with checking out. The advantage of this function is that other users can have access to the latest update while the current user can keep on editing.
5.5 Workflow and status management

Workflows represent the development process. They define the life cycle of a document, project, or process by specifying the sequences of statuses an object goes through the states. The workflow transition is defined as changing the status of a document version, which is also called promotion or demotion. When a document is promoted or demoted, workflow transition happens.

In SolidWorks Enterprise PDM, status is set to be an attribute of a version. Each status in a workflow represents a unique purpose. In figure 18, it is the editing interface of workflow. Red lines and blue boxes are the changing of status while yellow boxes are the statuses. Users are not allowed to edit the workflows except the super user. Super users can change properties of statuses.
In the theory developed by Henk Jan Pels, status should be ordered which means higher status code brings more usability than lower status code. Screenshot in figure 19 does not show ordered statuses, however, creating ordered statuses when editing workflows are possible and dependent on the users’ desires. In the workflow edition of SolidWork Enterprise PDM, the administrators are allowed to define the possible actions on a version in a certain state. In figure 19, the rights could be assigned to the states. By assigning increasing rights purposely, ordered state framework could be realized.

SolidWorks allows users to send notifications to specific parties including users themselves when there is a status change for one version or there is a checking in/out behavior. Users can communicate with each other by this notification to make their decision in project.

It is also recommended that status demotion should be avoided. In SolidWorks, the demotion is allowed. When constructing the workflow, users are allowed to create demotion possibilities.

5.6 Summary of SolidWorks Enterprise PDM and enhancements

Versioning in SolidWorks Enterprise PDM is time sequence based. It seems the complex versioning methods introduced in the scientific articles are not being appreciated due to their complexity and difficulties in being compatible with the object-oriented database. The simple time based versioning mechanism conforms to the reference model. Both versions and revisions in SolidWorks are ordered. In case there is a change of contents, version number or revision number change as well with new revision/version creation. In principle, the version coding in SolidWorks Enterprise PDM conforms to the reference model. The revision concept is not covered in the reference model. The revision in SolidWorks Enterprise PDM mixes the version and state since the revision is a kind of version for external reviewer except for allowing different revision numbers for the same content.

The revision concept in SolidWorks Enterprise PDM is functional and useful since it allows external reviewers to examine the contents of certain versions. The concept of revision is simple with only a number as its attribute. Revisions are just another set of numbers showing only the orders of improvement. Nevertheless, it is interesting to extend its concept and function to
classify the availability of the contents for different external reviewers. For example, if revision is not just a class with attribute number but with status as well, then the external reviewers could be classified as well to guarantee the validity of the contents that is supposed to be shown to a customer, supplier or a partner. A revision with status attribute differs nothing from a version except a revision is always for external reviewers outside the group developing this revision.

In figure 20, the organization and the external reviewers have different authorities to view contents of version 5. Version 5 is approved and assigned revision A with status Binnen, which means the at this moment revision A is only available for the organization developing it. Revision A could be promoted to status Buiten after its contents is recognized and validated by the management board in the organization. Status Buiten allows revision A to be examined by reviewers outside the organization. This extension can be applied to adapt the revisions to different reviewers with different accessibilities. By managing the status of a version or revision, visibilities for different reviewers can be controlled. In other words, improved status management enhances collaboration between different companies could be improved. When a revision is exposed to several companies, those companies are acknowledged at the same level.

The basic structure of the revision and status management could be illustrated in the UML diagram in figure 21. In figure 21, the revisions are open to specific reviewers. To which reviewers a revision is open depends on the review hierarchy. In the review hierarchy, reviewers’ name and code of hierarchy are stored. In business, the data of review hierarchy should be kept by the organization that develops the revision. In the promotion, the level in hierarchy promoted into should be recorded as well to correspond to the review hierarchy.
Similar extension could be applied to be adapted to different reviewers belonging to different parties. In figure 22, version 5 is designed to show the contents to both partners. The version 5 is approved and assigned revision A1 and revision B1. The point is that revision number consists of two elements. One element represents to whom the revision is shown. Revision A1 and Revision B1 have the same contents. In organization’s database, revision A1 is the synonym of revision B1. Version 6 evolves from version 5 and approved as well receiving revision number A2. The revision A2 is only used to show the contents to partner A while partner B should not be acknowledged about the contents of version 6. This revision number assignment mechanism does not bother the status of the revision. In the example, the revision A2 is at the same level with revision B1. The improvement is that the content of a version or a revision is adapted to different external partners.

The checking in/out mechanism in SolidWorks Enterprise PDM conforms to the reference model. The checked out version can not be edited by other users guaranteeing that a version
should not be edited by more than one user simultaneously. Nevertheless, a robust problem still exists that a checked out version could not be edited when it is not checked in timely. This problem is still waiting for adequate solution.

Status management in SolidWorks Enterprise PDM is roughly consistent with the reference model. Ordered set of status can be established by the administrators. Demotion is technically possible by setting up the workflow that allows the decreasing of status. However, it is not recommended to decrease the status. How is ordering, how is state linked to workflow?

### 6. Teamcenter Express

Teamcenter Express is the most popular PDM software package at present based on its large number of licensed users. The Teamcenter Express is quite different from SolidWorks Enterprise PDM in terms of data distribution, checking in/out, version and status management. The confusion is that the concept revision is totally different from the revision in SolidWorks Enterprise PDM. In Teamcenter Express, the revision is actually the version, which means there is no difference between version and revision in Teamcenter Express.

#### 6.1 Data distribution in Teamcenter Express

In Teamcenter Express, there is a distinction between product and document. The product in called item in Teamcenter Express, thus in our discussion we also use the term to represent product. In Teamcenter Express, item data is created by filling predefined forms. After checked in, the first revision of that item data is created as well. The figure 23 shows the basic item data model provided by Teamcenter.

![Figure 23 Basic Item Data Model](image)

By definition, item in Teamcenter Express is data globally applicable to all revisions of one item, while item revision is data applicable to one single revision of the item. The architecture of data distribution in Teamcenter Express is shown in figure 24. However, it is not the case at all that this rule could be generalized across other PLM software packages.
In figure 25, class Item has attributes shared by all its item revisions. When starting creating a new item, the itemID, Name and Rev are required to input in defined format itemID/Rev-Name. In this format, the name is not the value of the attribute in class Item; instead, it is just a description of the revision. The attribute information of an item or item revision is entered by item master or item revision master. Each item has only one item master form for the attribute information input, so does item revision.

The document in Teamcenter Express is called dataset. The architecture concerning dataset is shown in figure 25. The dataset class contains the identifier and revisionID as the attributes. The identifier and revision number are assigned to a dataset when the dataset is created. They indicate to which item or item revision the dataset corresponds. The document version is used to record the modification to the dataset. Therefore, the document in Teamcenter has both revisions and versions. The versioning of dataset discusses the version evolution of the documents.

### 6.2 Versioning in Teamcenter Express

The versioning of item revisions and document versions in Teamcenter is also time sequence based. The numbering of versions could be rather flexible since the revisions and versions are ordered. Example of numbering of revision is below.

- itemID/A, itemID/B, itemID/C,…..
- itemID/11, itemID/12, itemID/13,……
6.3 *Complex object versioning*

The complex objects in Teamcenter Express are versioned based on time sequence as well. The complex object revisions evolve when the complex object’s revision is checked out and then checked in. This way of creating new complex object revisions is similar to SolidWorks Enterprise PDM. The evolution of complex objects’ revisions do not cause evolution of the complex object’ revision.

In the Teamcenter interface, expanding a complex item exposes its constituent items. In figure 26, there is a complex object 0086536-Fuselage. It contains one constituent objects which is 0086539-radar. 0086538-radar is one of the components of 0086536-Fuselage and also it is one object listed in parallel with Fuselage as well. When new revisions of radar are created, no matter which one is revised (component or object), both are updated. The item itself and its shortcuts are all updated when any shortcut of the item or the item instance itself is updated. However, complex object Fuselage is not revised automatically when its components are revised. Therefore no new revision of Fuselage is created. With this phenomenon, it is clear that no static binding is applied in Teamcenter Express.

Figure 26 Item shortcut in Teamcenter
6.4 Checking in/out in Teamcenter Express

Checking in/out a version is used to create a new version of dataset in Teamcenter. A user needs to check an item out before he wants to start editing. Documents or items checked out are put in local working space. The location of checked out item or document in the local working space can be specified by the administrators.

An item or document has an attached checking history. The history saves the activities (checking in or checking out), the user (who does that) and date/time (when it is done). The UML diagram representing this function is below in figure 27.

Figure 27 Checking in/out history in Teamcenter

In Teamcenter, there are two checking in/out mechanisms, explicit and implicit. Explicit checking out is initiated when users use check in/out options. That means right click followed by check in and out options triggers explicit checking in/out. Implicit checking out cannot be initiated. It is automatic and occurs when dataset is opened for modification. After modified, dataset is automatically checked back in to database as well. A drawback of implicit checking is that the checking is not registered in the checking history.

6.5 Status management in Teamcenter Express

In Teamcenter Express, Status can be customized by the administrators. Both document and item have status as attribute. The values of this attribute are defined statuses. Figure 28 shows the interface of creating new status types. The authority of changing status is problematic. The authority of assigning statuses could be customized as well.
In Teamcenter interface, release statuses are graphically represented. Figure 29 is derived from URE’s application. The symbol 60 represents the indicated item is released after finishing workflow 60. The workflow 60 is one of the workflows in URE. The workflows in URE are named by tens, such as 10, 20, 30, etc. The symbol means the indicated item is released after finishing all the workflows. The symbol indicates that the indicated item is in certain storage inside a loop of workflows.

In status management in URE’s Teamcenter application, there are three predefined release statuses, 20, 60 and released (graphically represented as ). 20 is status of checking in URE. 10 is status of checking rejected. Once an item is assigned any one of the release statuses, checking it out will be impeded while only read access is possible.
6.6 Summary of Teamcenter Express

Teamcenter Express applies time sequence based versioning for both items and documents. Similar to the versioning in SolidWorks Enterprise PDM, Teamcenter Express allows much freedom in assigning codes to documents and revisions.

The check in/out mechanism in Teamcenter Express consists of implicit checking and explicit checking. The division of checking in/out mechanism does not bother the theoretical term; instead, it concerns only operating the software. The checking in/out mechanism protects the document or product by prohibiting editing on a checked out object, which is the same as SolidWorks Enterprise PDM and obeys the theoretical model as well. In implicit checking in/out, the operation is not recorded in the history. This could be a little problem since it deteriorates the modification record.

The statuses in Teamcenter Express are also an ordered set. Higher status codes represent higher maturity of the contents of a document or product. However, there is no evidence proving the workflow statuses could be edited with hierarchy in action allowance (higher statuses do not allow more actions necessarily).

7. Problem definition reflection

7.1 Versioning in PLM software packages

The versioning in both software packages (SolidWorks Enterprise PDM and Teamcenter Express) are based on time sequence. While the Teamcenter Express is based on item versioning with BOM application by allowing edition on the configuration of a product, the SolidWorks Enterprise PDM is document based. There is no real BOM application in the SolidWorks Enterprise PDM.

Both software packages do not support the tracing of the version evolution. On one hand, the simple time sequence versioning casts a shadow on the version evolution, which means the time sequence versioning itself does not show the evolution at all. On the other hand, there is no such a function in either software package designed to trace the evolution of versions. There could be several reasons for the lacking of this function. First, the advantage of tracing version evolution is not significant in facilitating designing and production. Currently, both software packages highlight the latest version only in order to simply indicate which version should be used to create a new version or for production. Thus, it is not meaningful to bother about other branches in the version evolution. Second, the theories of tracing version evolution (usually known as version derivation graph (VDG)) are far from practical application due to the complexity in practical version management. The versioning in practice are usually network versioning instead of linear. Then composing and showing a network is still not easy to manipulate and understand. Therefore, there is no enough reason for the two software packages to apply any version derivation function.
7.2 Checking in/out mechanism

Checking in/out mechanisms is similar in both software packages and conforms to the theoretical models. In order to edit on an existing version of a document or product, the user has to check it out into a local workspace (could be personal computer or local file vault). Before checking in, the modified content is already saved, but not published yet. Both software packages provide a protection for the checked-out document or product by forbidding other users to check it out. There are different functions to check in/out in two software packages; however, they do not affect the conformation to the theoretical models.

7.3 Status management

In SolidWorks Enterprise PDM, the status is called state. In order to clarify the confusion, in this chapter we take the naming in most of the scientific articles by calling status instead of state. The status management in two software packages varies a lot. In general, the status management in SolidWorks Enterprise PDM better fits the theoretical model by facilitating all the rules. In Teamcenter Express, there are ordered codes for status, but there is no interface to edit the rights of the users on a version in a certain status. Therefore, status management in SolidWorks Enterprise PDM is better based on the criteria of theoretical models.

8. Conclusion

The versioning mechanisms in the selected software packages have something in common while keeping individually specific properties. Teamcenter Express focuses more on the version management within the organization while SolidWorks Enterprise PDM takes care of the collaboration among organizations by offering the concept revision (revision in SolidWorks Enterprise PDM) besides managing organizational data. The architectures of versioning in two software packages both have their advantages and shortcomings, nevertheless, some performance allowed by software packages violate the theoretical models.

Teamcenter Express uses version for document and revision for product. SolidWorks Enterprise PDM has the concepts of version and revision both for document since there is no product data in SolidWorks. Versioning mechanisms in both software packages are based on time sequence. The versioning methods suggested in literatures are not preferred. The complex object versioning in both software packages are based on time sequence as well. In order to create a new version of a complex object, the complex object itself (product) should be checked out and then checked in. Any modification to its constituent parts does not create a new version of this complex object.

The statuses in both software packages are ordered sets. In SolidWorks Enterprise PDM, higher statuses could represent more actions allowed if the statuses are edited adequately. Teamcenter Express does not provide this function. Therefore, SolidWorks Enterprise PDM better fits the theoretical model of status management. The revision in SolidWorks Enterprise PDM represents a version with a special status used to collaborate with external partners. Therefore, the concept
of revision mixes with the status, which is not validated by current theories. The suggested theory extension advises to extend the status management to collaborate the organizations.

The checked out document or product are both protected by allowing only read-only access in both packages. The checking in/out in Teamcenter Express contains implicit and explicit checking. The implicit checking in/out allows the users to change the contents of a version without evolving it. That should be avoided in application since it violates the reference model seriously.

The versioning mechanisms in the two software packages partly obey the reference models while violate some important rules. The violations are still being applied in current business; therefore, there is still a long way ahead to guide the software packages to obey the theoretical rules and to improve the theories in the mean time.
References


- [Suzanne, 1951] Briet, Suzanne, “Qu'est-ce que la documentation?”, 1951